LED 流水灯实验及仿真

1 实验简介

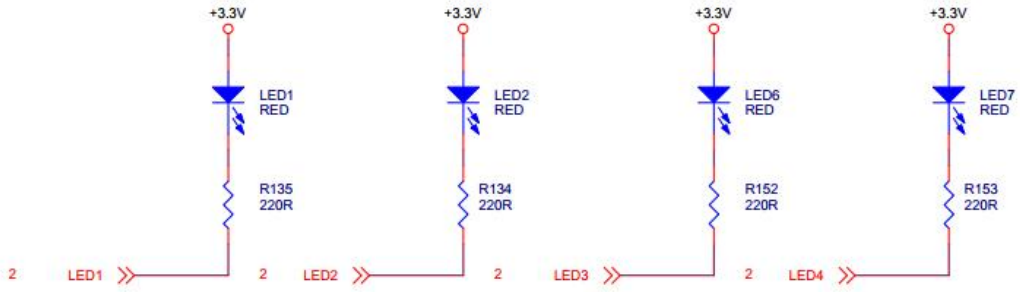
通过 LED 流水灯实验，介绍使用 PDS 软件开发 FPGA 的基本流程，器件选择、设置、代码编写、编译、分配管脚、下载、程序 FLASH 固化、擦除等；同时也检验板上 LED 灯是否正常。

2 实验环境

- Windows 10 64 位
- PDS
- AXP100 开发板

3 实验原理

3.1 LED 硬件电路



LED 部分原理图

从上面的 LED 部分原理图可以看出，开发板都是将 IO 经过一个电阻和 LED 串联接电源端，FPGA 的 IO 输出低电平点亮 LED。IO 输出高电平 LED 灯熄灭，其中的串联电阻都是为了限制电流。

3.2 程序设计

FPGA 的设计中通常使用计数器来计时，对于 200Mhz 的系统时钟，一个时钟周期是 5ns，那么表示一秒需要 200000000 个时钟周期，如果一个时钟周期计数器累加一次，那么计数器从 0 到 199999999 正好是 200000000 个周期，就是 1 秒的时钟。。

程序中定义了一个 32 位的计数器：

```
//Define the time counter
reg [31:0]      timer;
```

最大可以表示 4294967295，十六进制就是 FFFFFFFF，如果计数器到最大值，可以表示 21.474836475 秒。程序设计中是每隔 0.25 秒 LED 变化一次，一共消耗 4 秒做一个循环。

```
always@(posedge sys_clk or negedge rst_n)
begin
    if (~rst_n)
            timer <= 32'd0;
    else if (timer == 32'd199_999_999)
            timer <= 32'd0;
    else
            timer <= timer + 1'b1;
end
```

在 0.25 秒、第 0.5 秒、0.75 秒、1 秒到来的时候分别改变 LED 的状态，其他时候都保持原来的值不变。

```
// LED control
always@(posedge sys_clk or negedge rst_n)
begin
    if (~rst_n)
            led <= 4'b0000;
    else if (timer == 32'd49_999_999)
            led <= 4'b0001;
    else if (timer == 32'd99_999_999)
            led <= 4'b0010;
    else if (timer == 32'd149_999_999)
            led <= 4'b0100;
```

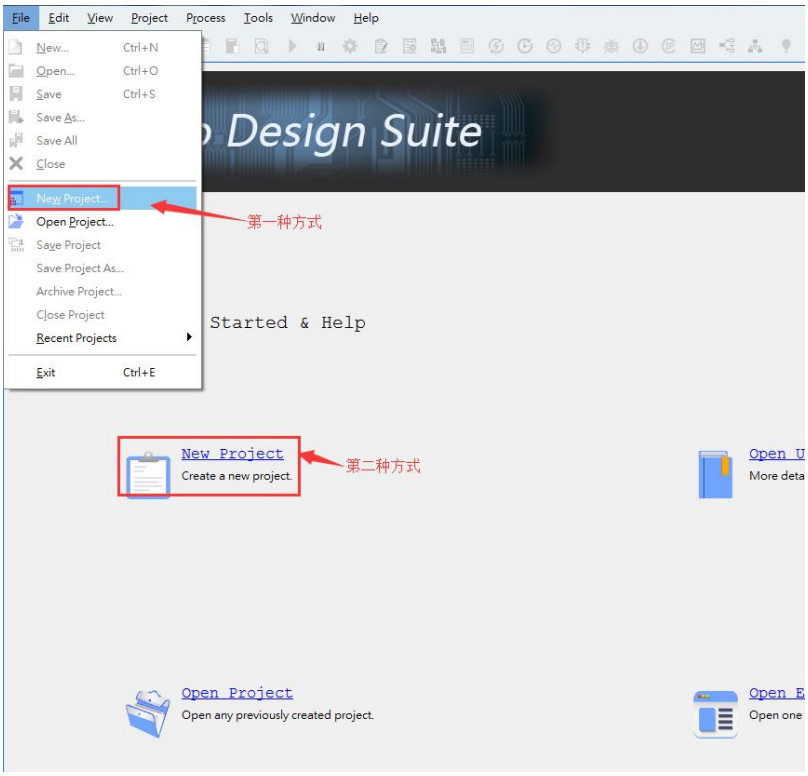
```
        else if (timer == 32'd199_999_999)
                led <= 4'b1000;
    end
```

4 PDS 工程

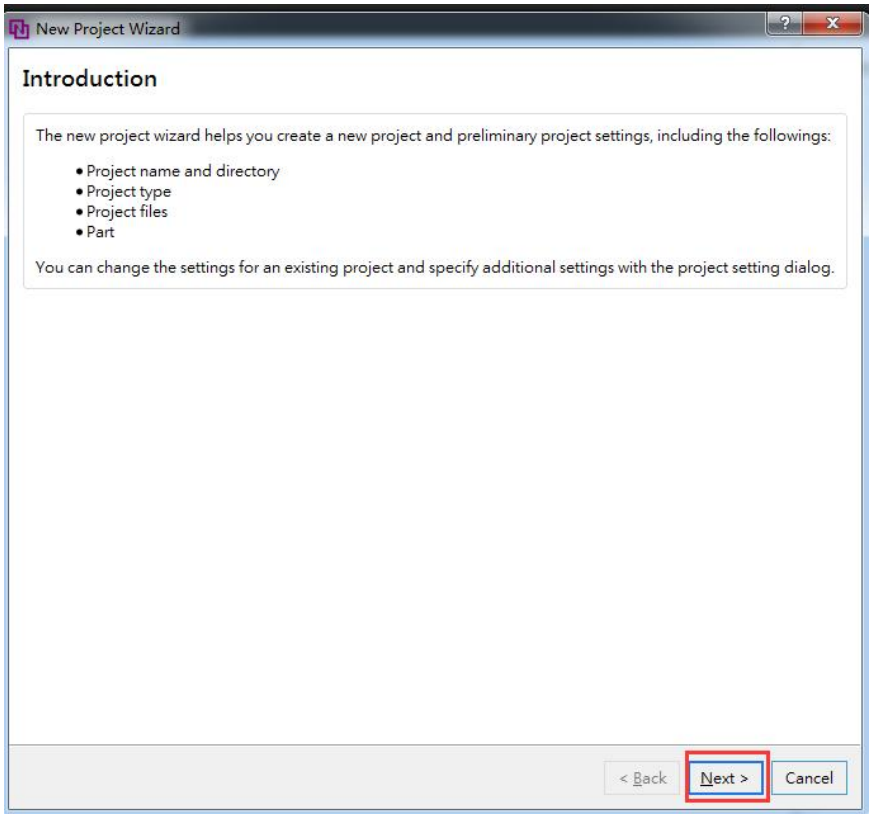
4.1 创建工程

1. 启动 Pango Design Suite 2021.1-sp7 开发环境(在开始菜单中选择 pango->Pango Design Suite 2021.1-sp7>Pango Design Suite 。Pango Design Suite（简称 PDS）或者双击桌面的 Pango Design Suite 2021.1-sp7 的图标直接打开软件。

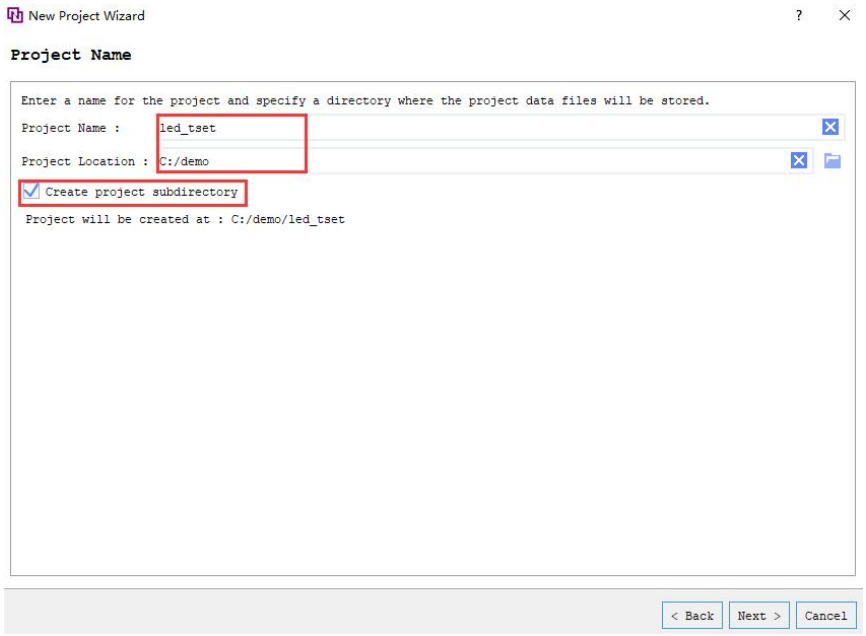
2. 在 PDS 开发环境里双击 Create Project 或 File->New Project...这两种方式都可，如下图:



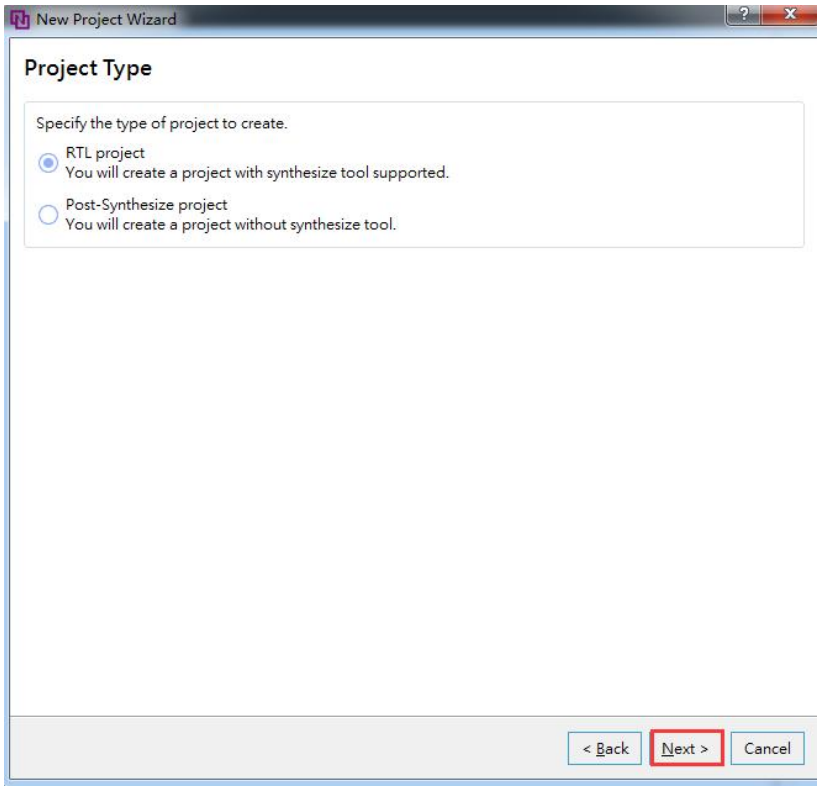
3. 弹出一个 PDS 的工程向导，点击 Next 按钮。



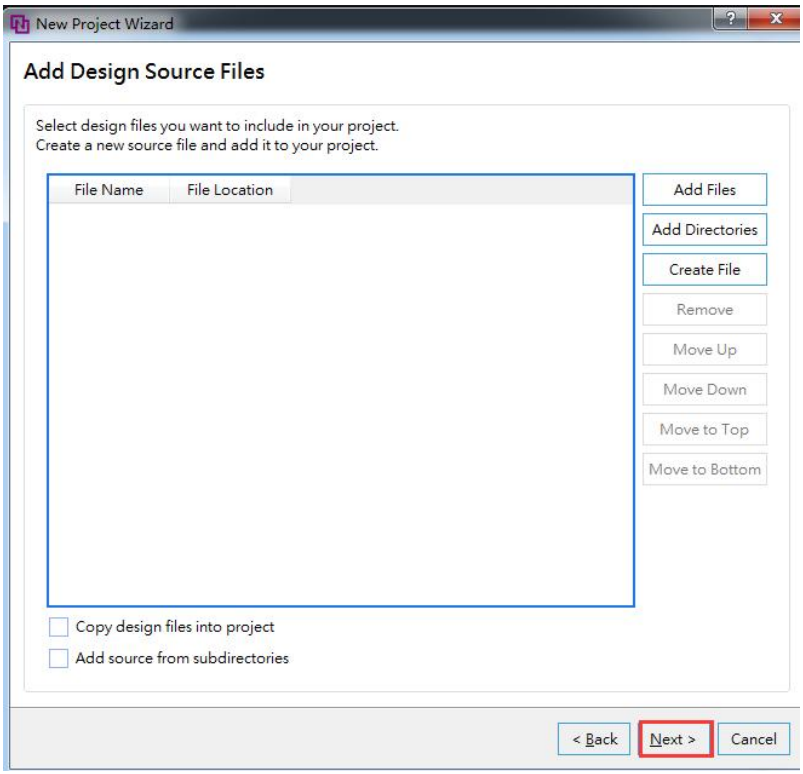
4. 在弹出的对话框中输入工程名和工程存放的目录，这里取一个 led_test 的工程名，点击 Next；



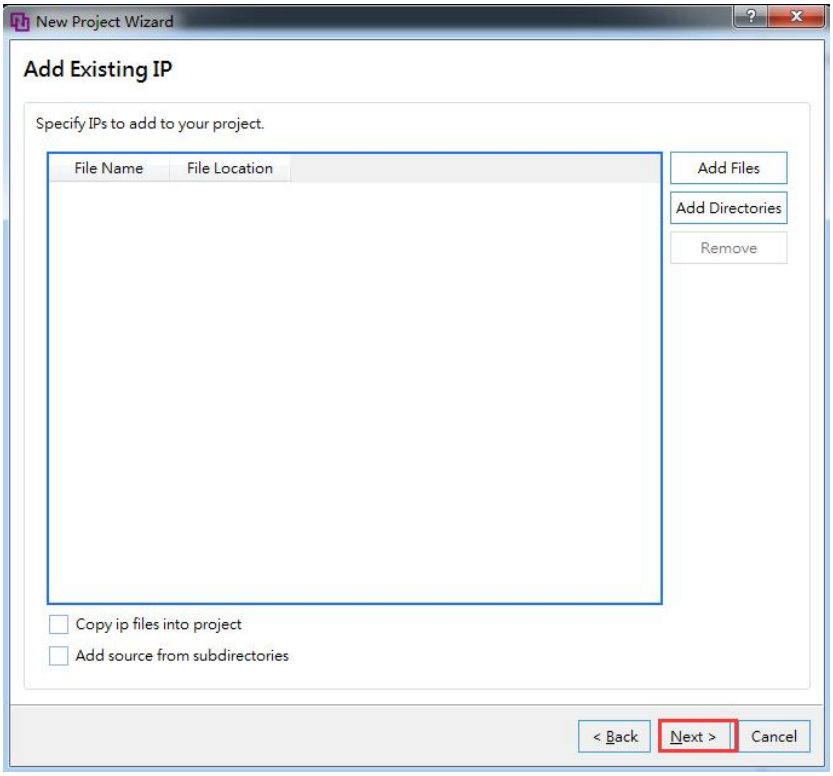
5. 在下面的对话框中默认选择 RTL Project, 因为我们这里使用 verilog 行为描述语言来编程，单击 Next



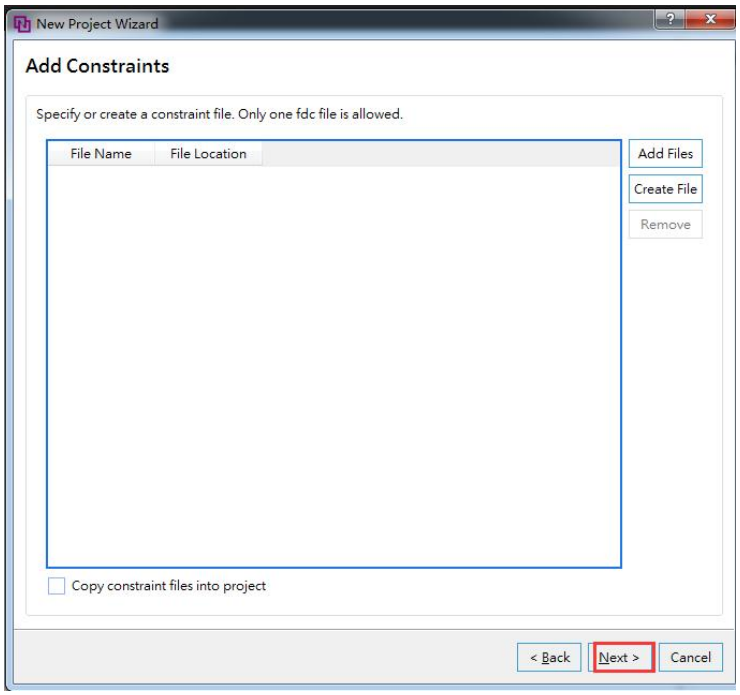
6. 进入 Add Design Source Files 界面，这里先不添加任何设计文件。点击 Next；



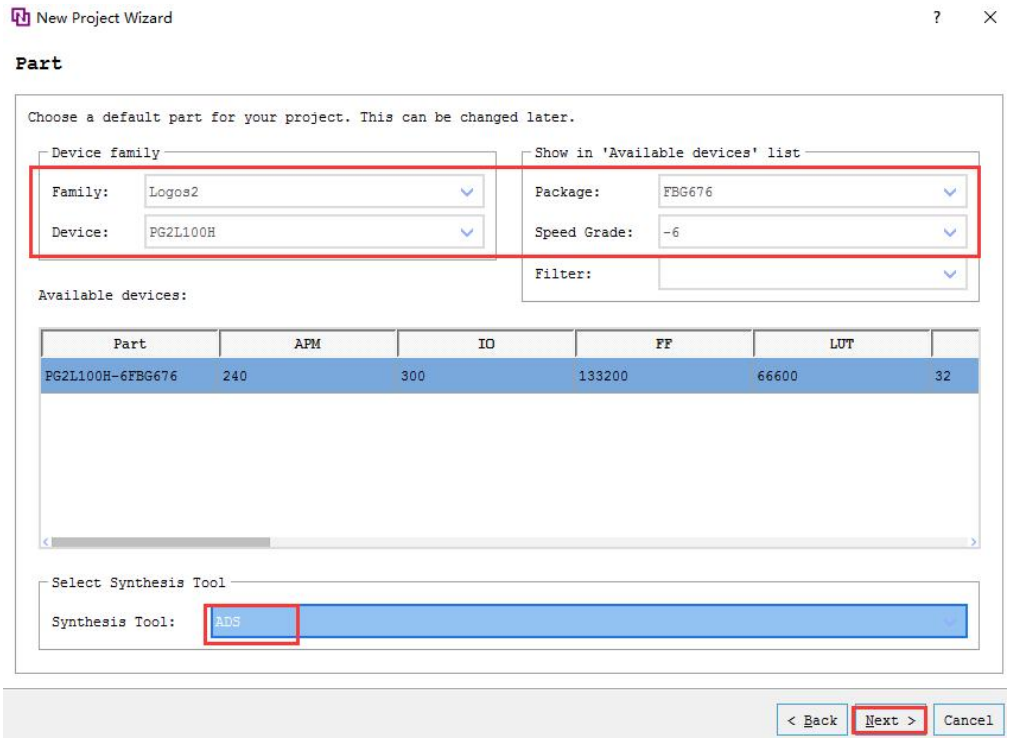
7. 这里问是否添加已有的 IP，保持默认不添加，单击 Next；



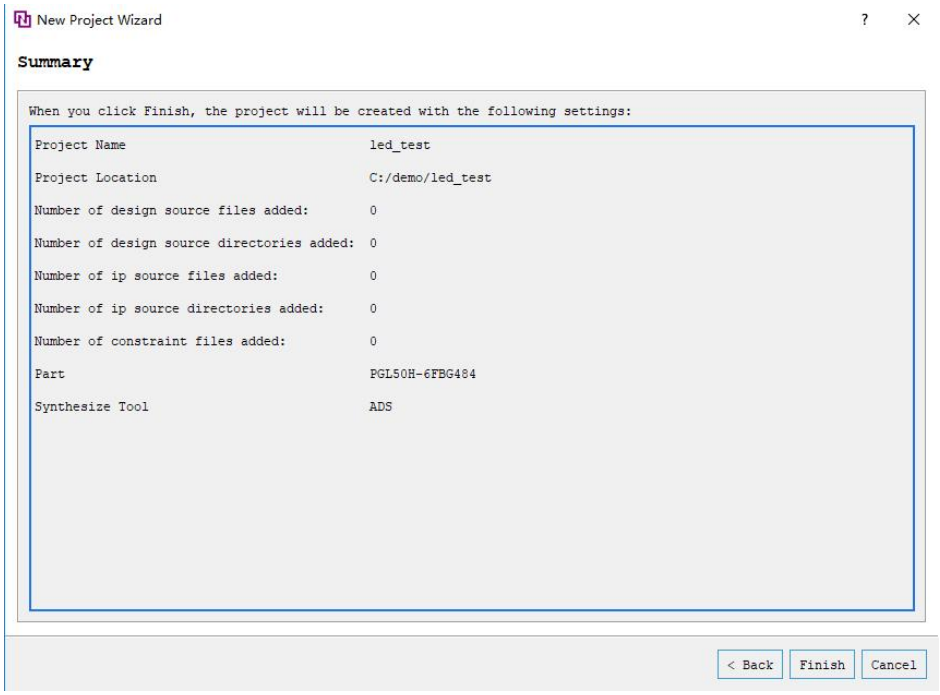
8. 提示是否添加已有的约束文件，这里约束文件我们也没有设计好，也不添加。



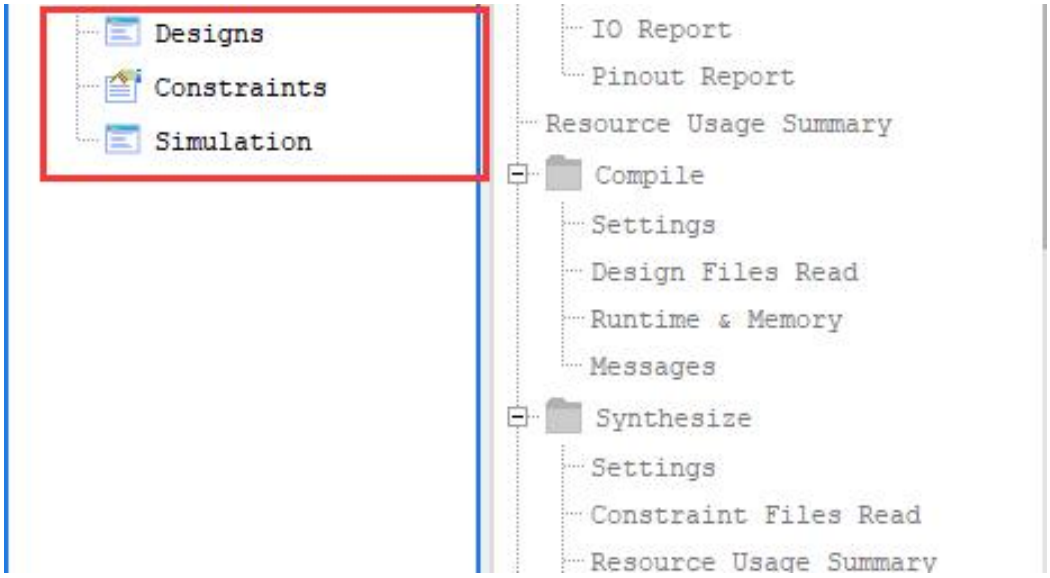
9. 在接下来的对话框选择所用的 FPGA 器件，以及进行一些配置。首先在 Family 栏里选择 Logos2，Device 中选择 PG2L100H，在 Package 栏选择 FBG676, Speed grade 栏选择-6；综合工具选择 ADS；单击 NEXT 进入下一界面：



10. 再次确认一下板子型号有没有选对, 没有问题再点击“Finish”完成工程创建。

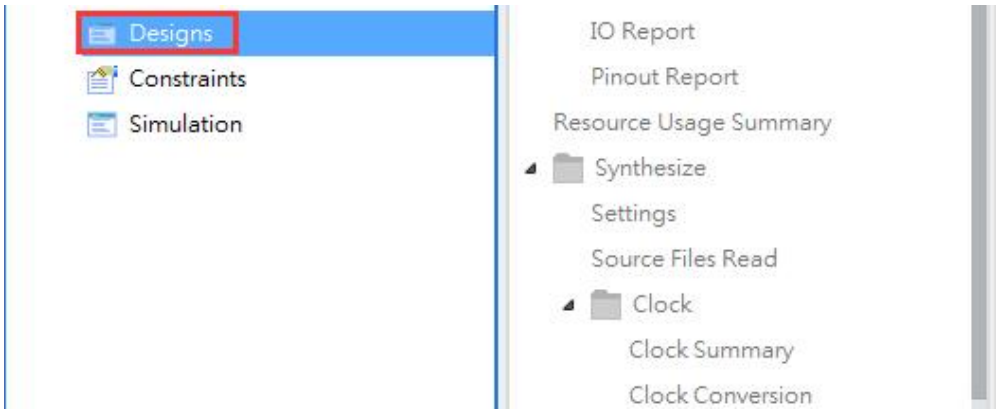


11. 工程创建后如下图所示：

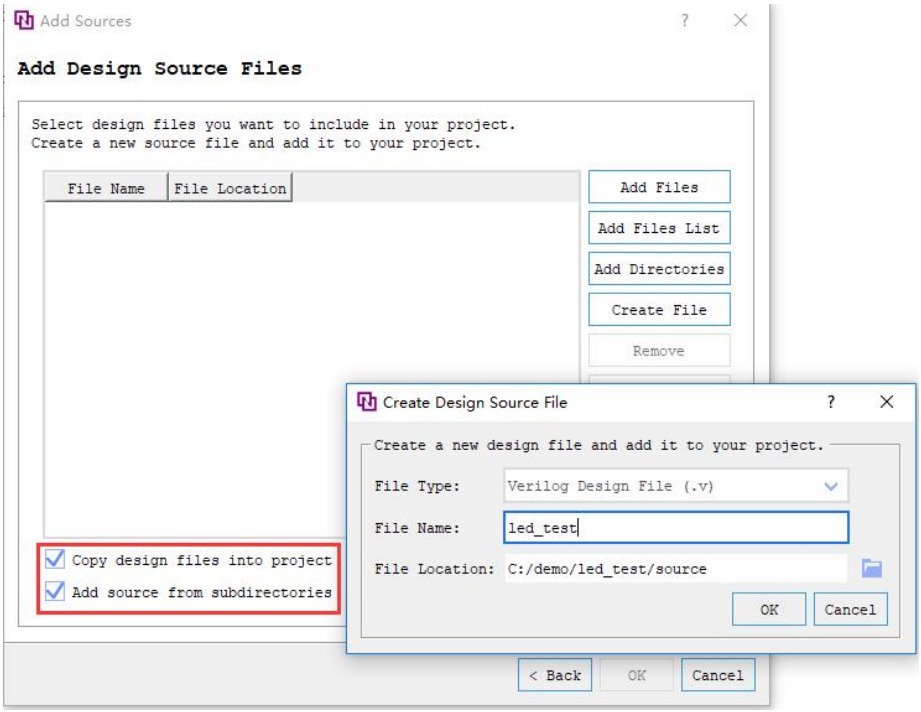


4.2 编写流水灯的 verilog 代码

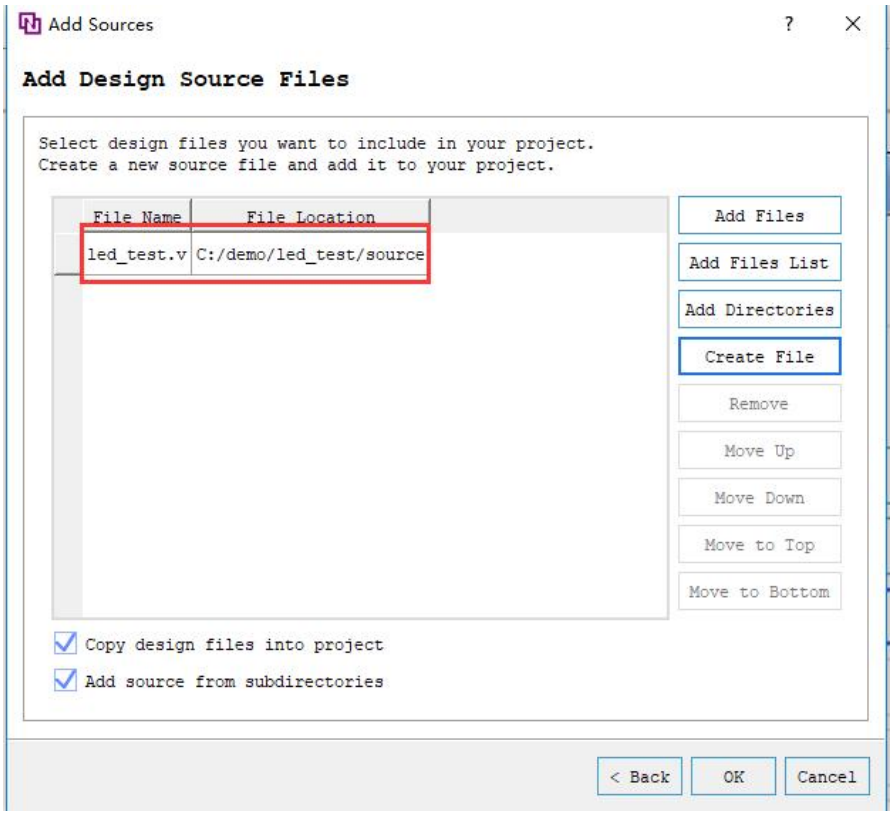
1. 双击 Sources 下的 Designs 图标;



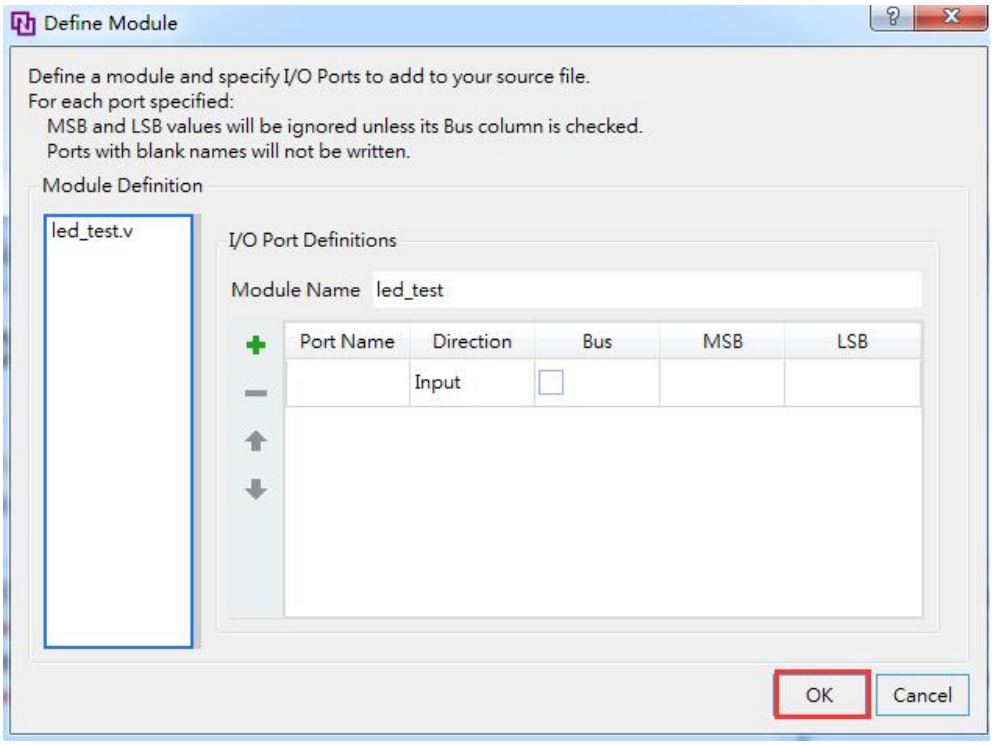
2. 在 Add Design Source Files 界面中进行如下设置，点击 OK；



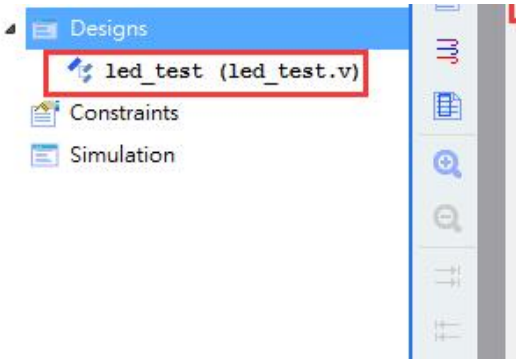
3. 可以看到已经新建发 led_test.文件，点击 OK 按钮。



向导会提示您定义 I/O 的端口，这里我们可以不定义，后面自己在程序中编写就可以,单击 OK 完成。



这时在 Navigator 界面下的 Designs 里已经有了一个 led_test.v 文件, 并且自动成为项目的顶层（Top）模块了。



4. 接下去我们来编写 led_test.v 的程序,这里我们定义了一个 32 位的寄存器 timer, 用于循环计数 0~199_999_999(4 秒钟), 当计数到 49_999_999(0.25 秒）的时候，熄灭第一个 LED 灯；当计数到 99_999_999(0.5 秒）的时候，熄灭第二个 LED 灯；当计数到 149_999_999(0.75 秒）的时候，熄灭第三个 LED 灯；当计数到 199_999_999(1 秒）的时候，熄灭第四个 LED 灯，计数器再重新计数。具体的操作直接看代码吧。

```
        `timescale 1ns/1ns
module led_test
(
input        sys_clk_p,        //system clock positive
input        sys_clk_n,        //system clock negative
input        rst_n,           //reset ,low active
output       led_test,
output reg[3:0]   led    // LED,use for control the LED signal on board
 );

wire sys_clk;
//define the time counter
reg [31:0]   timer;
assign fan_pwm=1'b0;
assign led_test=1'b1;
GTP_INBUFGDS sys_clk_ibufgds
(
.I                  (sys_clk_p            ),
.IB                 (sys_clk_n            ),
.O                  (sys_clk              )
);

//////////////////////////////////////////////////////////////////////////////
// cycle counter:from 0 to 1sec
//////////////////////////////////////////////////////////////////////////////
always @(posedge sys_clk or negedge rst_n)
begin
 if (~rst_n)
    timer <= 32'd0;                // when the reset signal valid,time counter clearing
 else if (timer == 32'd199_999_999)   //4 seconds count(200M-1=199999999)
    timer <= 32'd0;                 //count done,clearing the time counter
 else
```

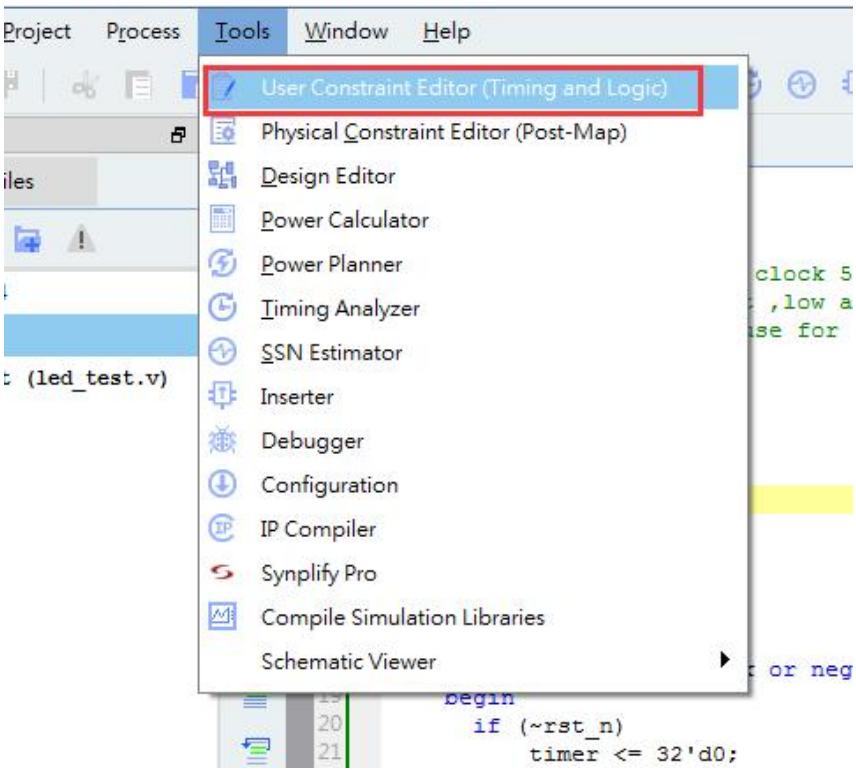
```
    timer <= timer + 1'b1;          //timer counter = timer counter + 1
end
//////////////////////////////////////////////////////////////////////////////
// LED control
//////////////////////////////////////////////////////////////////////////////
always @(posedge sys_clk or negedge rst_n)
begin
 if (~rst_n)
    led <= 4'b0000;               //when the reset signal active
 else if (timer == 32'd49_999_999)   //time counter count to 0.25 sec,LED1 lighten
    led <= 4'b0001;
 else if (timer == 32'd99_999_999)   //time counter count to 0.5 sec,LED2lighten
 begin
    led <= 4'b0010;
  end
 else if (timer == 32'd149_999_999)   //time counter count to 0.75 sec,LED3 lighten
    led <= 4'b0100;
 else if (timer == 32'd199_999_999)   //time counter count to 1 sec,LED4 lighten
    led <= 4'b1000;
end
endmodule
```

5. 编写好代码后保存,点击菜单 File -Save All。

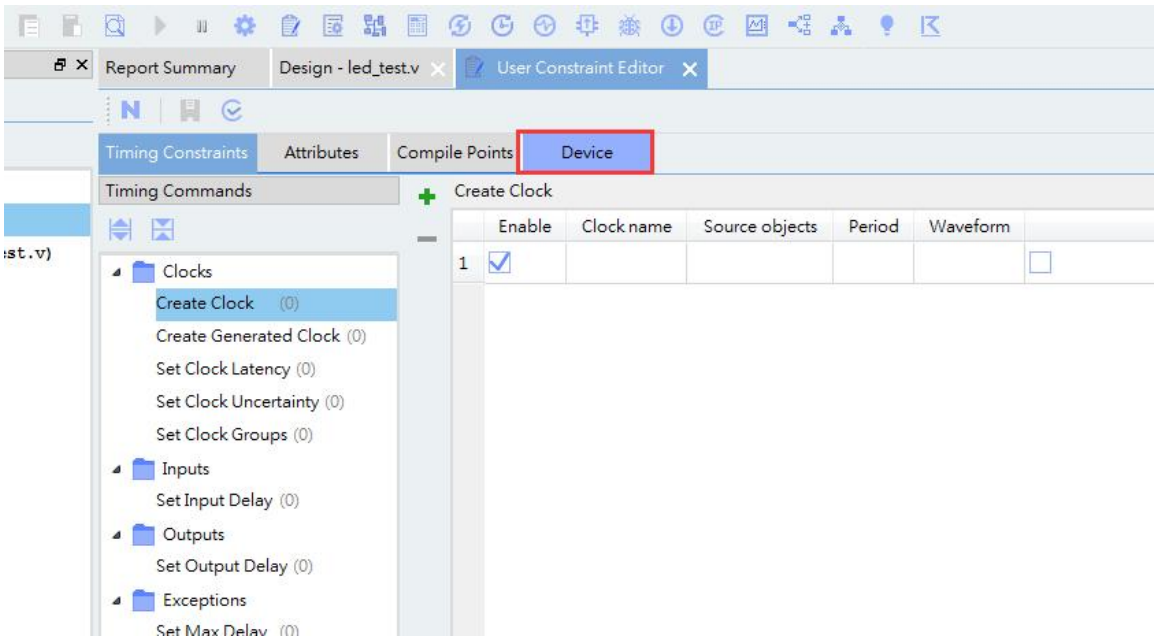
4.3 添加 UCE 约束

User Constraint Editor(Timing and Logic)简称 UCE，主要是完成管脚的约束,时钟的约束,以及组的约束。这里我们需要对 led_test.v 程序中的输入输出端口分配到 FPGA 的真实管脚上。

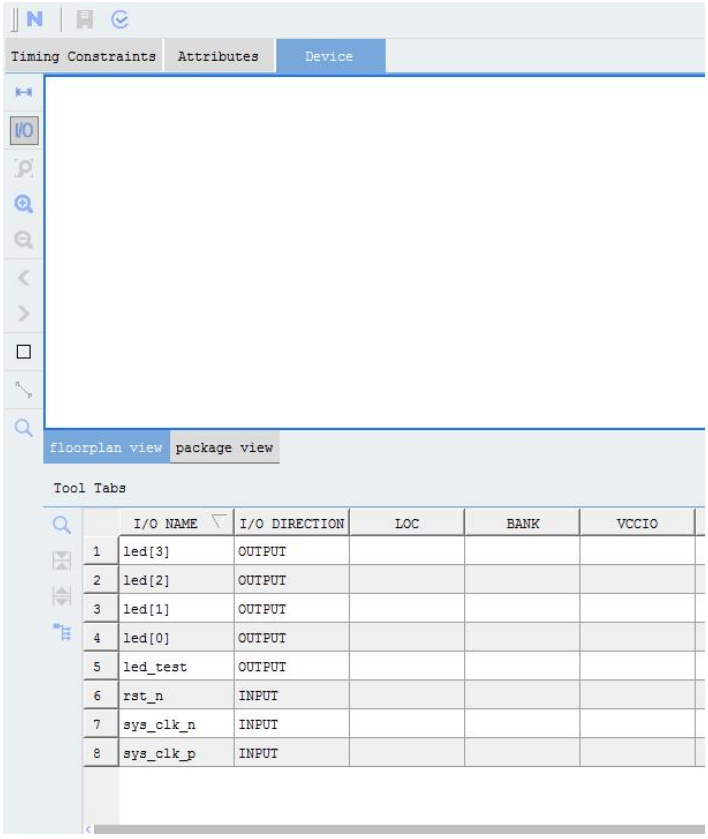
1. 击菜单栏“Tools”下的"User Constraint Editor";



2. 在弹出的界面中单击 Device;

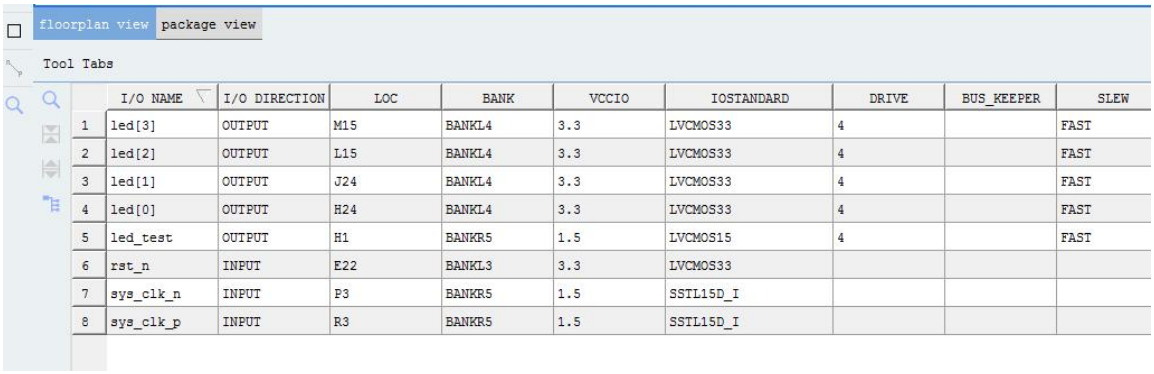


3. 在 Device 中单击 I/O，可看到工程中用到的 IO 端口;



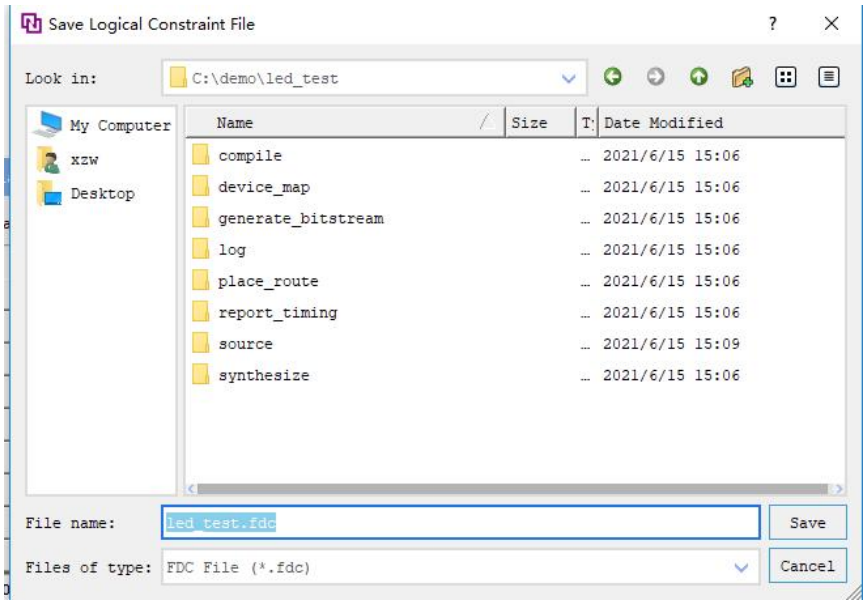
	I/O NAME	I/O DIRECTION	LOC	BANK	VCCIO
1	led[3]	OUTPUT			
2	led[2]	OUTPUT			
3	led[1]	OUTPUT			
4	led[0]	OUTPUT			
5	led_test	OUTPUT			
6	rst_n	INPUT			
7	sys_clk_n	INPUT			
8	sys_clk_p	INPUT			

4. 按如下方式分配管脚，LOC 就是与硬件中 FPGA 相对应的管脚，VCCIO 是 FPGA 的 IO 的电压标准，与硬件对应，其它在这里保持默认即可；



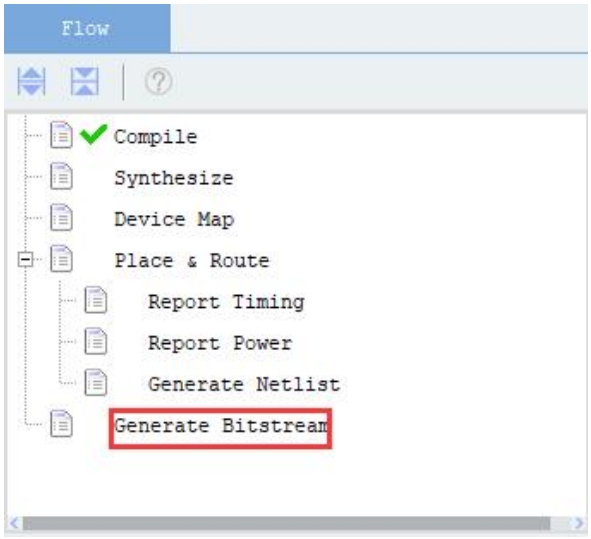
	I/O NAME	I/O DIRECTION	LOC	BANK	VCCIO	IOSTANDARD	DRIVE	BUS_KEEPER	SLEW
1	led[3]	OUTPUT	M15	BANKL4	3.3	LVCMOS33	4		FAST
2	led[2]	OUTPUT	L15	BANKL4	3.3	LVCMOS33	4		FAST
3	led[1]	OUTPUT	J24	BANKL4	3.3	LVCMOS33	4		FAST
4	led[0]	OUTPUT	H24	BANKL4	3.3	LVCMOS33	4		FAST
5	led_test	OUTPUT	H1	BANKR5	1.5	LVCMOS15	4		FAST
6	rst_n	INPUT	E22	BANKL3	3.3	LVCMOS33			
7	sys_clk_n	INPUT	P3	BANKR5	1.5	SSTL15D_I			
8	sys_clk_p	INPUT	R3	BANKR5	1.5	SSTL15D_I			

5. 单击保存后会弹对话框，在这里选择默认；

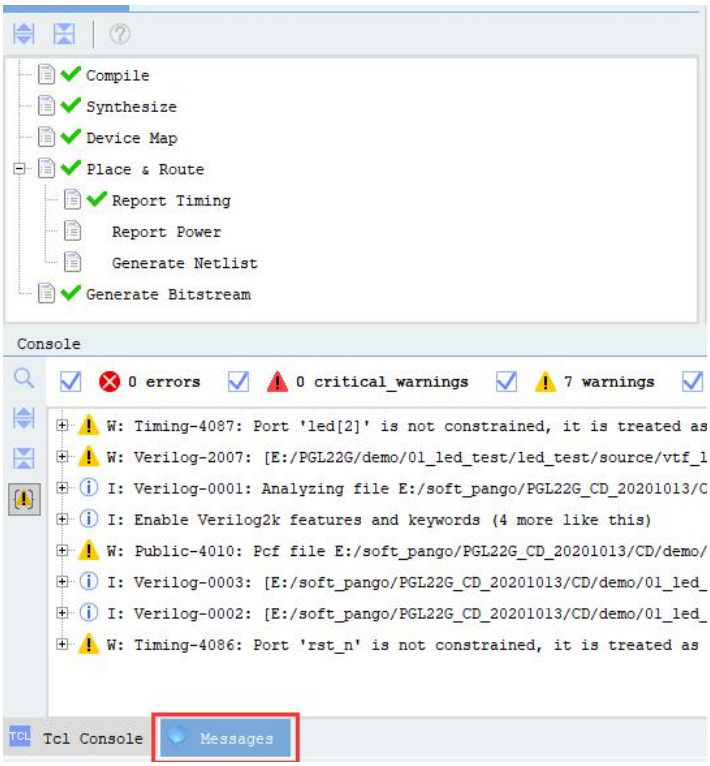


4.4 生成位流文件

双击 Generate Bitstream，然后软件会按照 Synthesize-> Device Map-> Place & Route-> Generate Bitstream 来产生位流文件。



如果工程在生成位流文件过程中没有错误，则会出现下图中每一步都正确的“√”，否则就会在 Messages 栏中显示 errors 的错误。



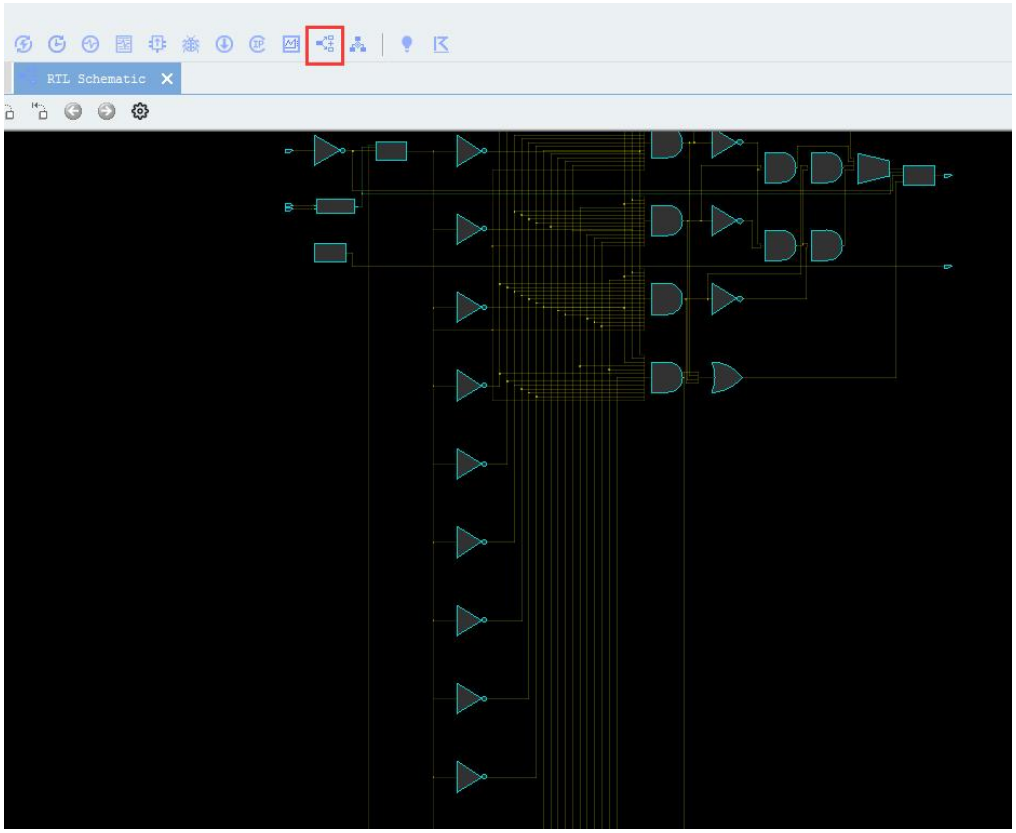
位流文件生成完成后，我们可以在 Report Summary 页面的到了 FPGA 资源的使用情况。

Resource Usage Summary

Find:

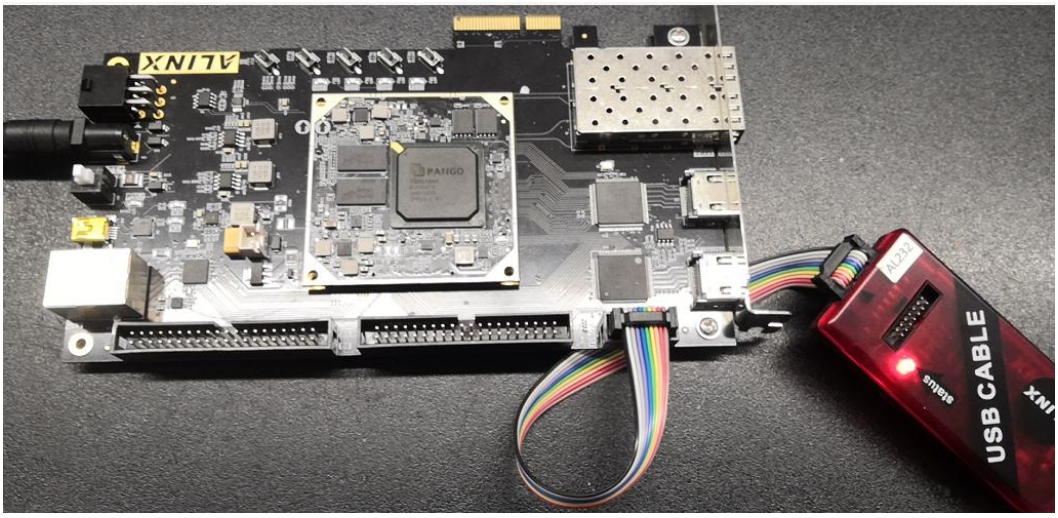
Logic Utilization	Used	Available	Utilization(%)
Use of MRCKB	0	12	0
Use of MFG_TEST	0	1	0
Use of KEYRAM	0	1	0
Use of IOLHR	7	300	3
Use of IOCKMUX_TEST	0	6	0
Use of IOCKB	0	24	0
Use of IO	8	300	3
IOBS	5	156	4
IOBD	3	144	3
Use of HSSTLP	0	2	0
Use of HCKMUX_TEST	0	8	0
Use of HCKB	1	96	2
Use of HARD0	10	10550	1
Use of GSEB	0	218	0
Use of GPLL	0	6	0
Use of DRM	0	155	0
Use of DDR_PHY	0	24	0
Use of DDRPHY_IOCLK_DIV	0	6	0
Use of DDRPHY_CPD	0	12	0
Use of CLMS	4	4975	1
LUT-FF pairs	4	19900	1
LUT	13	19900	1
FF	4	39800	1
Distributed RAM	0	19900	0
Use of CLMA	11	11675	1
LUT-FF pairs	32	46700	1
LUT	38	46700	1
FF	32	93400	1
Use of CCS	1	1	100
Use of BKCL	3	6	50
Use of APM	0	240	0
Use of ANALOG	0	1	0
Use of ADC	0	1	0

此外还可以通过下图操作查看 RTL 视图；

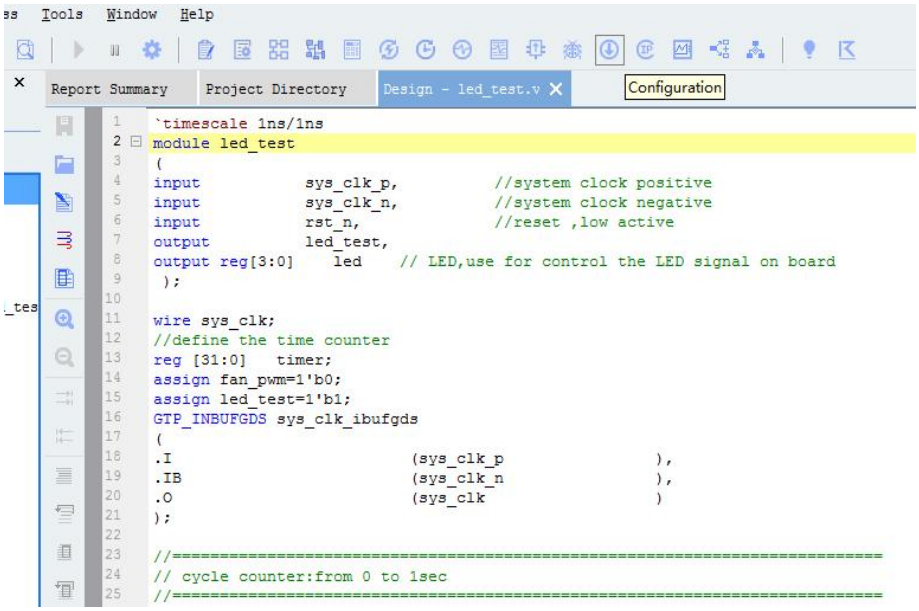


4.5 下载和调试

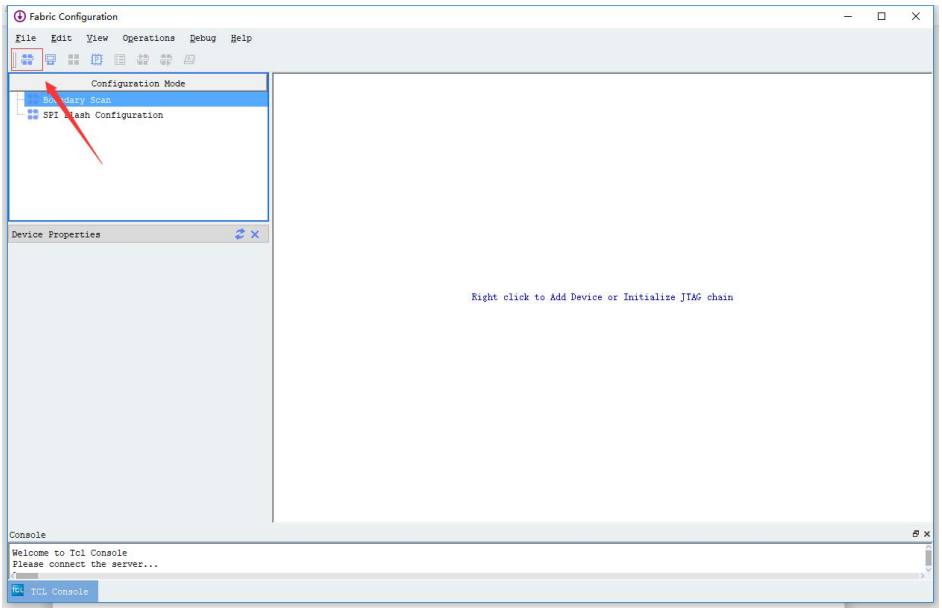
在上面生成了位流文件(.sbit)后，我们可以把 sbit 文件下载到 FPGA 芯片中，看一下 LED 实际运行的效果。下载和调试之前先连接硬件，把 JTAG 下载器和开发板连接，然后开发板上电，开发板的硬件连接图如下：



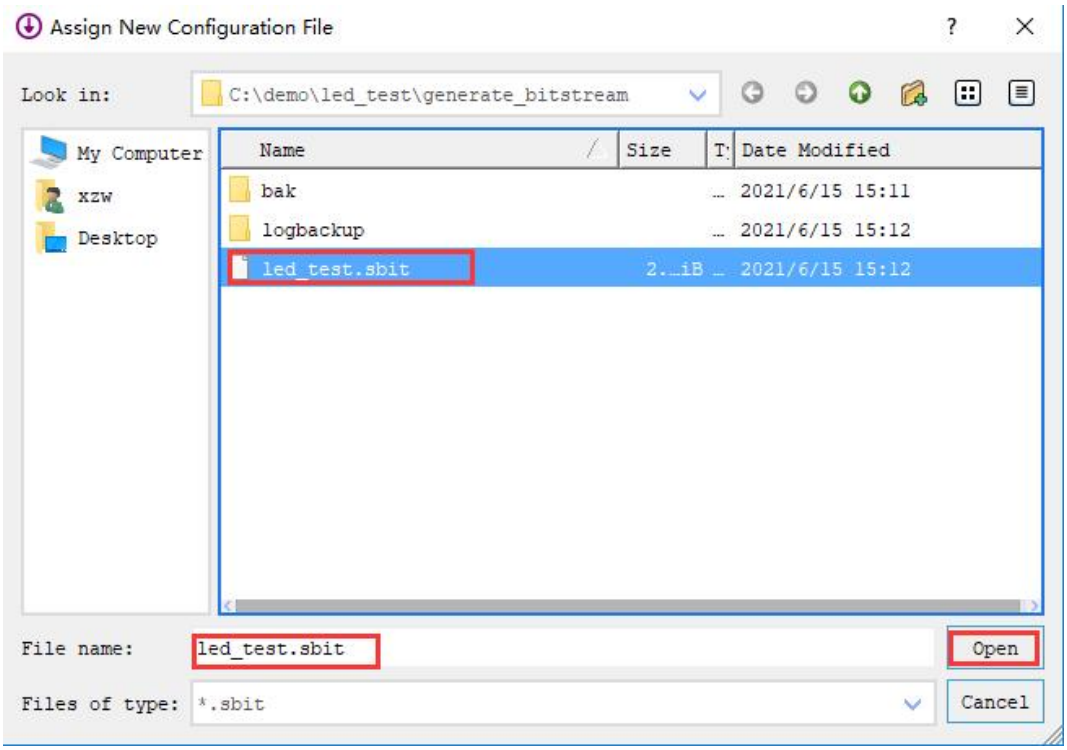
1. 单击界面中的“Configuration”按钮，作用一是下载程序到 FPGA 中运行；二是固化程序到 flash 中。



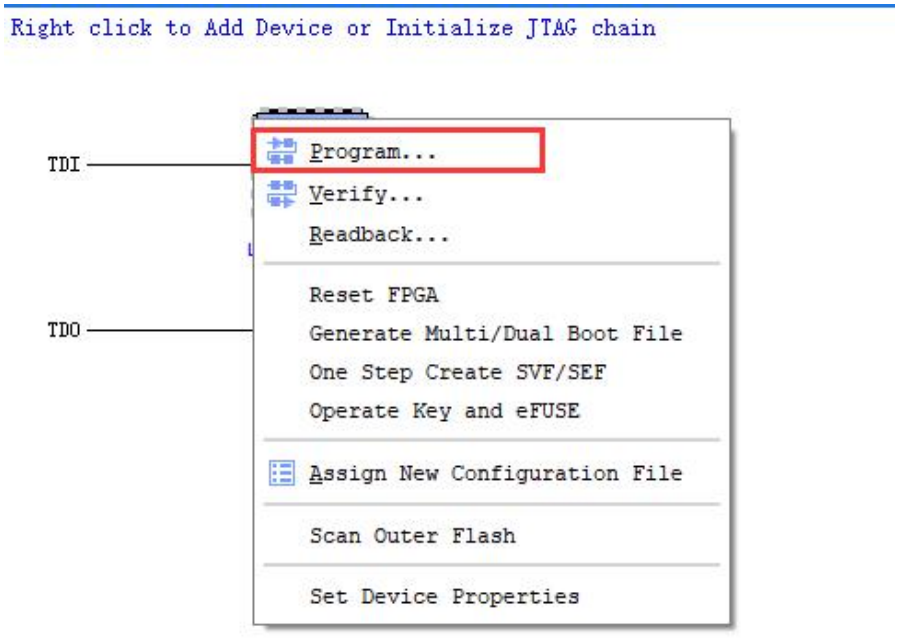
2. 在弹出的界面中的单击“Boundary Scan”，然后在右侧空白区单击右键选择“Scan Device”；



3. 在扫描到 JTAG 设备后会弹出如下对话框，并按如下加载.sbit 文件即可；



4. 然后可以看到左侧显示了要加载的文件，选中右侧绿色的方块，右击会弹出下拉菜单并选择"Program..."，下载完成后在板上可以在开发板上看到 LED 流水灯的效果。注意：这种方式程序是在 FPGA 运行，掉电后会消失。

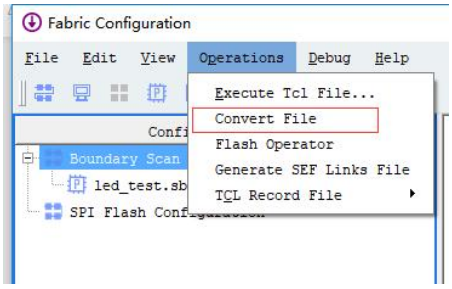


4.6 FLASH 程序固化

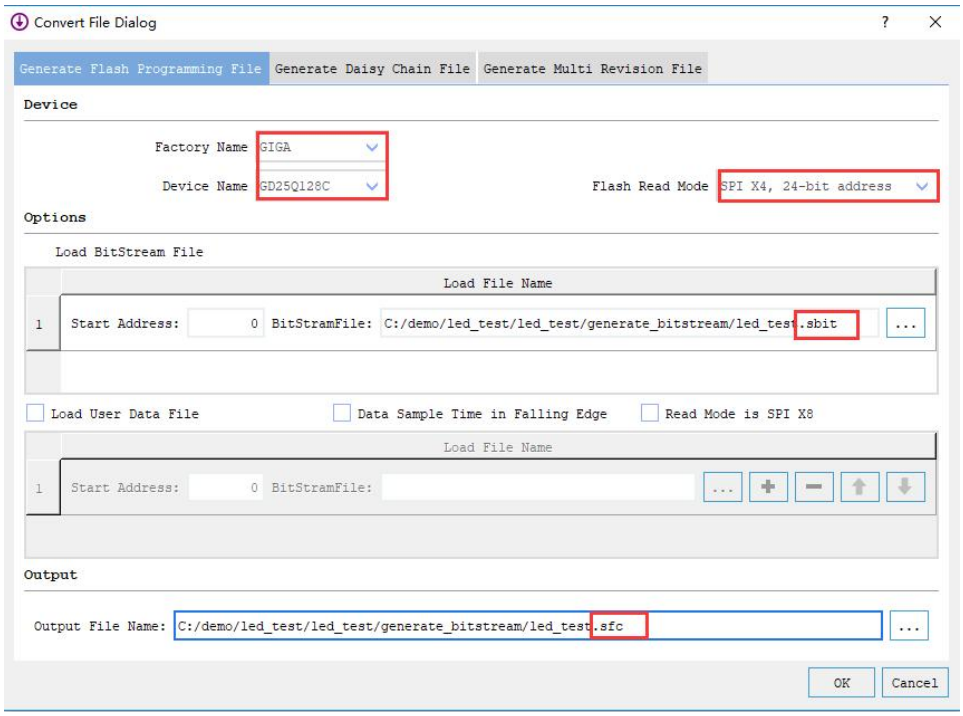
可能已经有朋友发现下载.sbit 文件到 FPGA 后，开发板重新上电后配置程序已经丢失，还需要 JTAG 下载。这岂不麻烦！好吧，这一节我们来介绍如何把配置程序固化到开发板上的 FLASH 中，这样不用担心掉电后程序丢失了。

在我们的开发板上有一个 8Pin 的 128Mbit 的 FLASH, 用于存储配置程序。我们不能直接把 sbit 文件下载到这个 FLASH 中，只能下载 sfc 文件到 flash 中。下面为大家介绍 FLASH 程序的固化的流程。

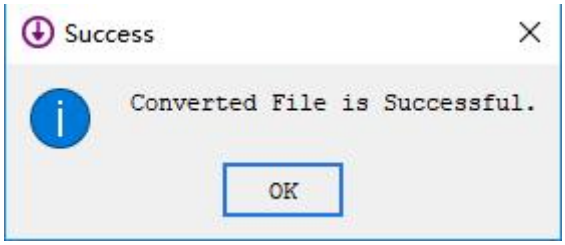
1. 首先，需要 sbit 文件转换成能下载的 flash 的 sfc 文件。在完成上节下载和调试后，选择菜单"Operations"下"Convert File"进行文件转换。



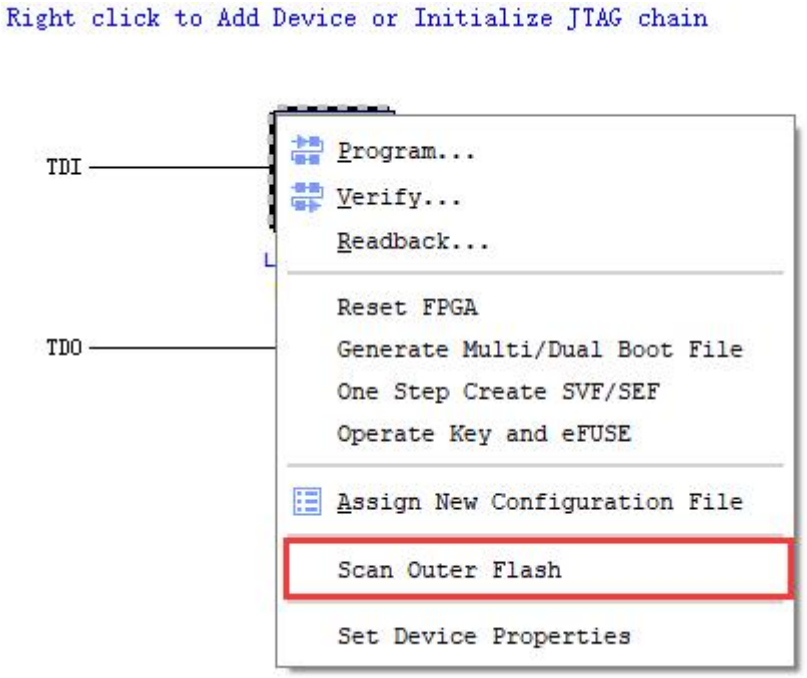
然后弹出如下界面，这里要根据硬件的 flash 型号来选择 flash 的厂家和设备型号，开发板实际用到的是 GD 的 GD25Q127，界面这里选择 GD25Q128C，程序兼容。Flash Read Mode 选择 SPI X4 然后选择要转换的 sbit 文件，点击 OK 即可转换；



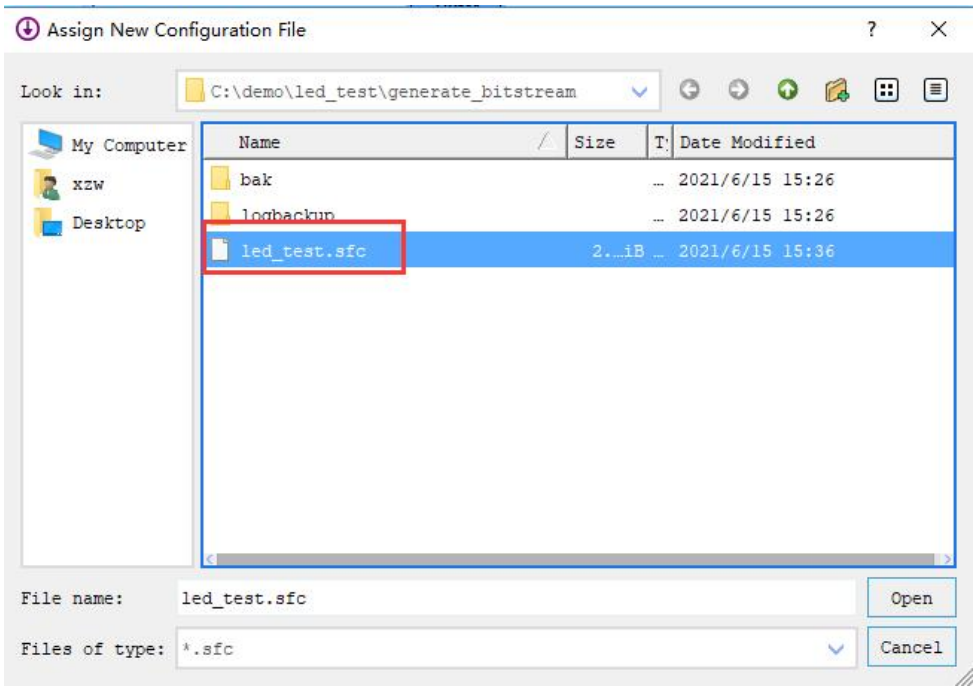
转换完成后显示如下界面，单击 OK；



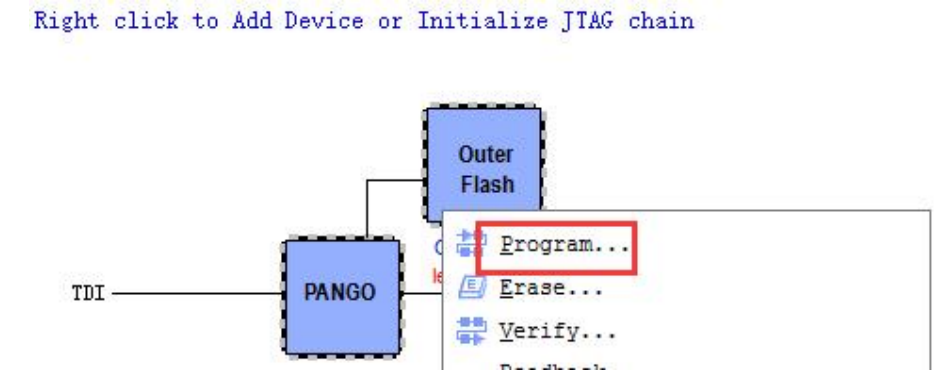
2. 选中右侧绿色的方块，右击会弹出下拉菜单并选择"Scan outer Flash"。



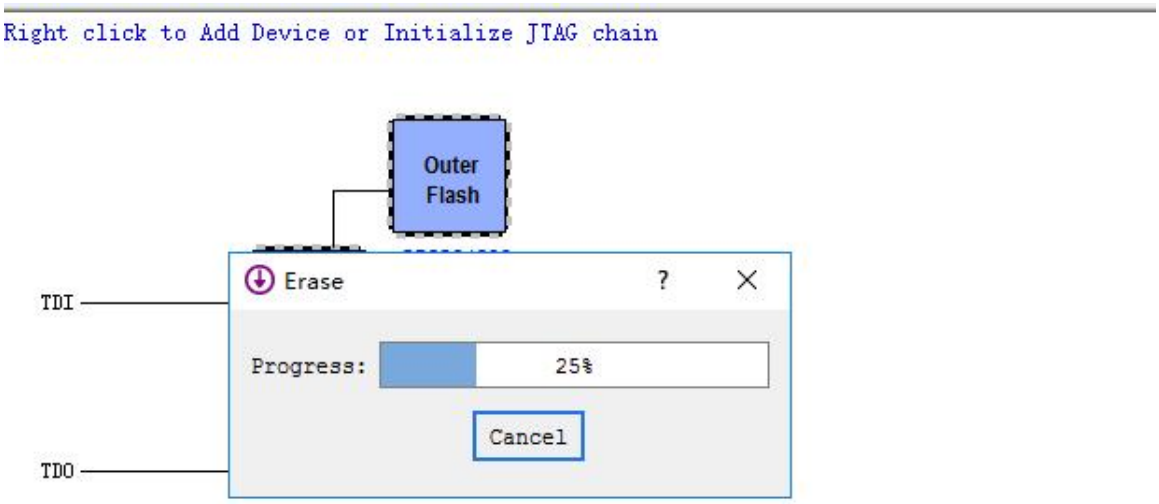
选择已生成的 sfc 文件，单击 Open；



可以看到界面中有了 flash 器件，选中“Outer Flash”绿色方块并右击选择菜单中“Program...”



弹出正在编程的进度界面，flash 编程完成后进度界面自动消失。

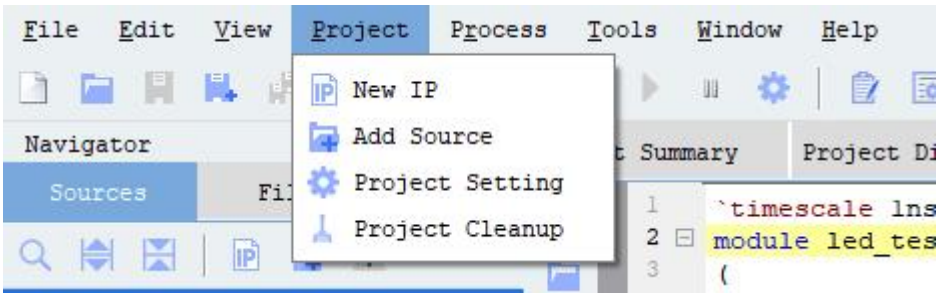


至此，SPI FLASH 烧写完毕，led_test 程序已经固化到 SPI FLASH 中了。我们来验证一下，关电重新启动开发板，等待一会儿你就可以看到开发板上的 LED 灯已经在做跑马运动了。

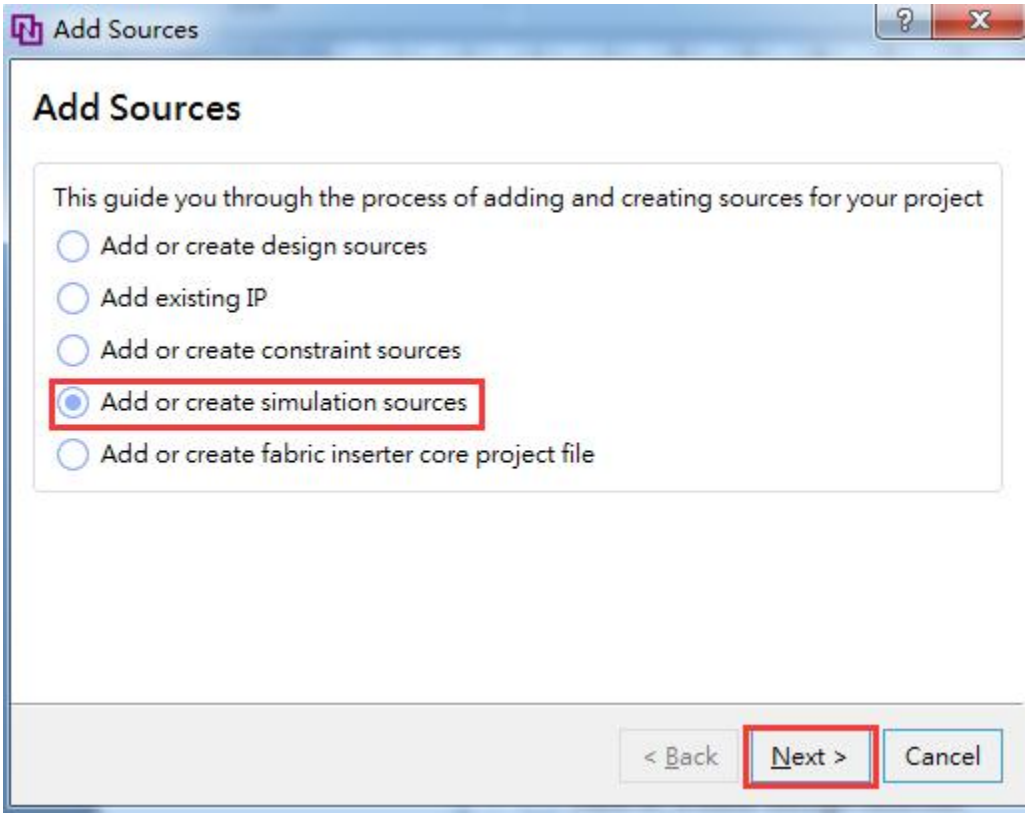
4.7 仿真验证

接下来我们不妨小试牛刀，让仿真工具 modelsim 来输出波形验证流水灯程序设计结果和我们的预想是否一致。具体步骤如下：

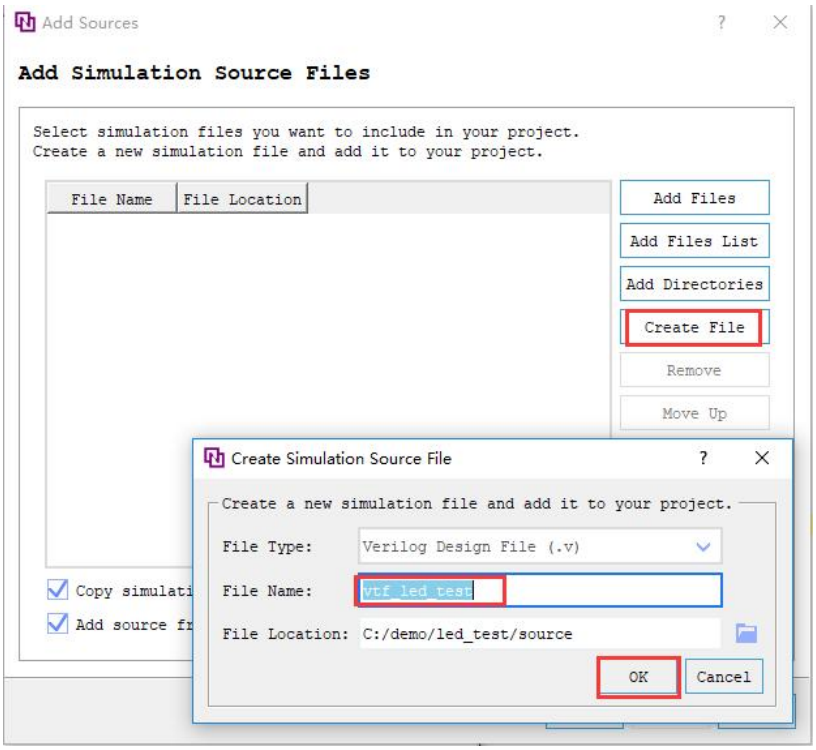
1. 添加激励测试文件，点击 Project 下的 Add Source；



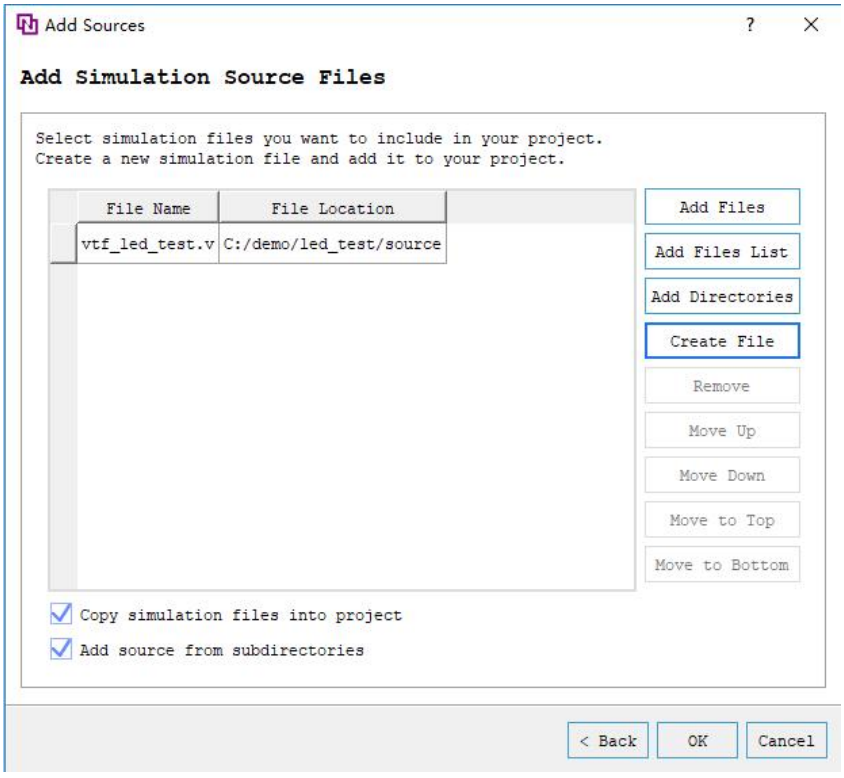
2. 点击 Add or create simulation sources 并"Next";



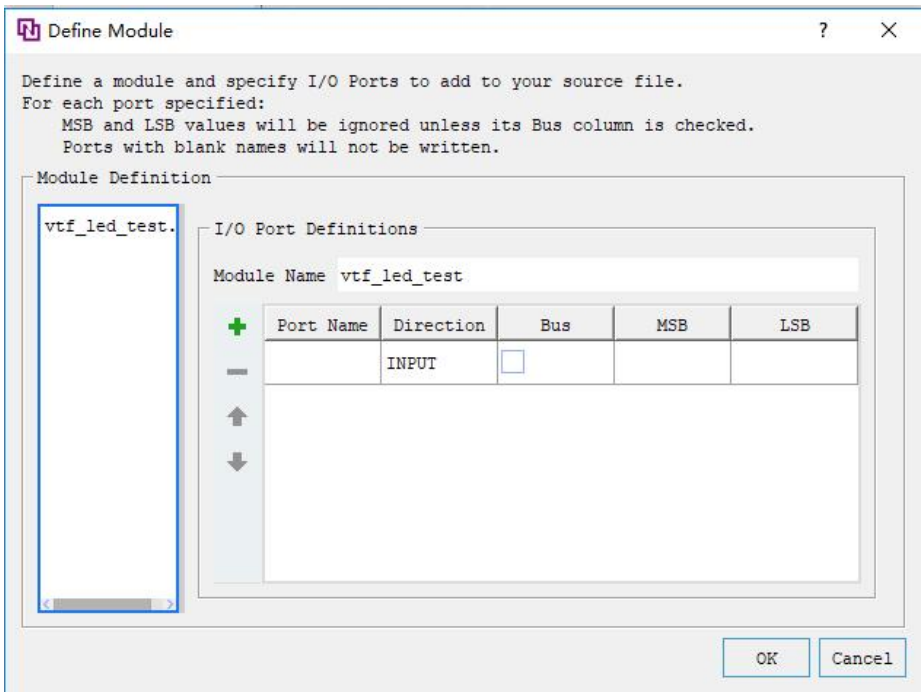
3. 在弹出的对话框中输入激励文件的名字，这里我们输入名为 vtf_led_test,其它按下图设置；



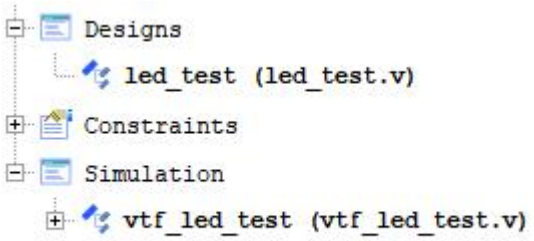
4. 点击 OK 按钮返回。



5. 这里我们先不添加 IO Ports，点击 OK。



6. 在 Simulation 目录下多了一个刚才添加的 vtf_led_test 文件。双击打开这个文件，可以看到里面只有 module 名的定义，其它都没有。

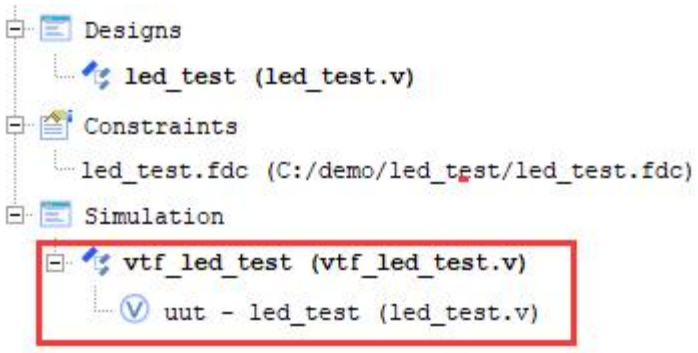


7. 接下去我们需要编写这个 vtf_led_test.v 文件的内容。首先定义输入和输出信号，然后需要实例化 led_test 模块，让 led_test 程序作为本测试程序的一部分。再添加复位和时钟的激励。完成后的 vtf_led_test.v 文件如下：

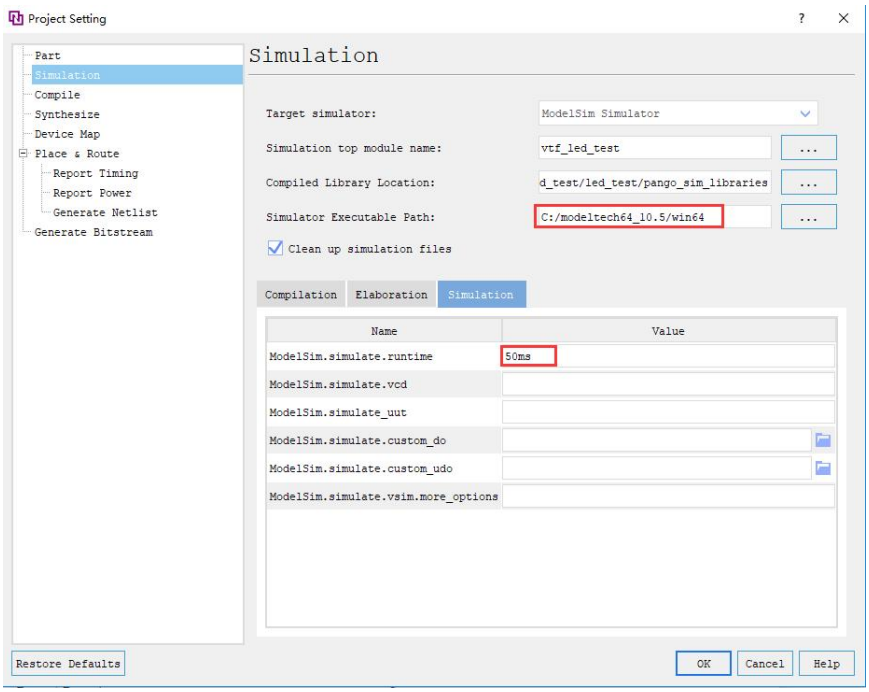
```
        `timescale 100ps / 100ps
module vtf_led_test;
  // Inputs
  reg sys_clk_p;
  wire sys_clk_n;
  reg rst_n;

  // Outputs
  wire [3:0] led;

  // Instantiate the Unit Under Test (UUT)
  led_test uut (
     .sys_clk_p(sys_clk_p),
     .sys_clk_n(sys_clk_n),
```

```
    .rst_n(rst_n),
    .led(led)
);

initial begin
    // Initialize Inputs
    sys_clk_p = 0;
    rst_n = 0;

    // Wait 100 ns for global reset to finish
    #1000;
      rst_n = 1;
    // Add stimulus here
    #20000;
  //  $stop;
 end

always #25 sys_clk_p = ~ sys_clk_p;
assign sys_clk_n=~sys_clk_p;

      endmodule
```

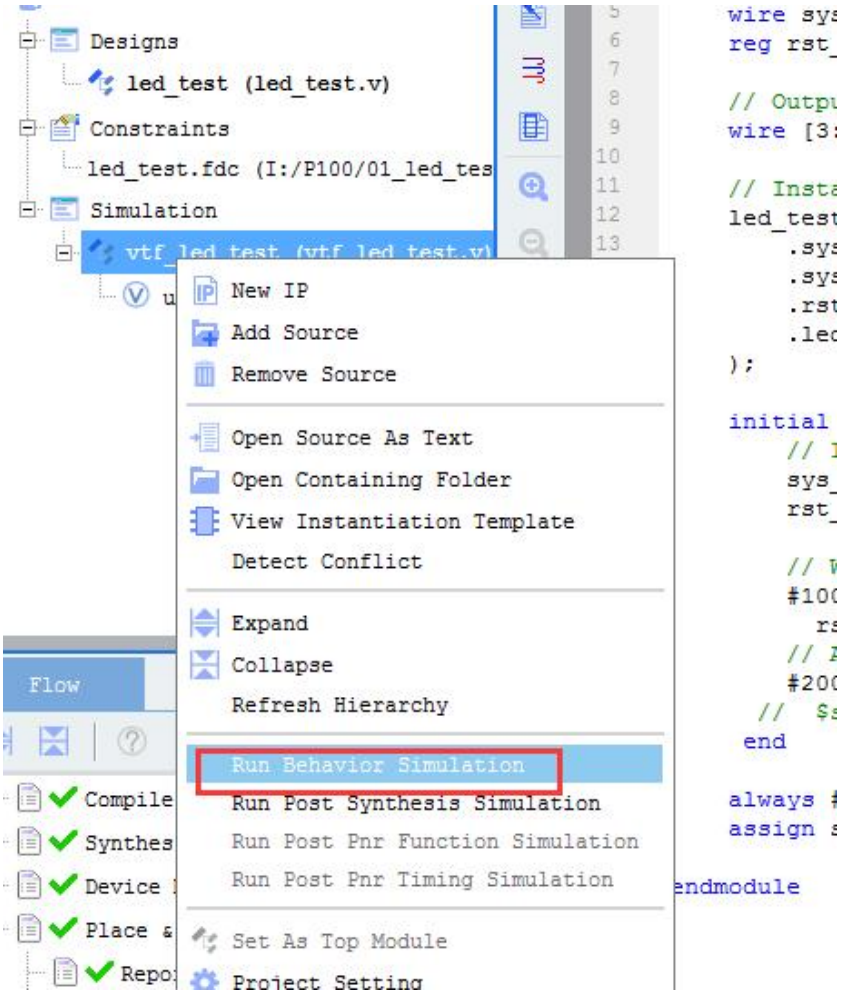
8. 编写好后保存，vtf_led_test.v 自动成了这个仿真的顶层了，它下面是设计文件 led_test.v；



9. 接下来设置 PDS 的仿真配置，在软件菜单 Project->Project Setting，然后在弹出的界面中进行如下设置，注意仿真库的路径在《Pango Design Suite 2021.1-sp7 安装》教程中已介绍。设置好后单击 OK。

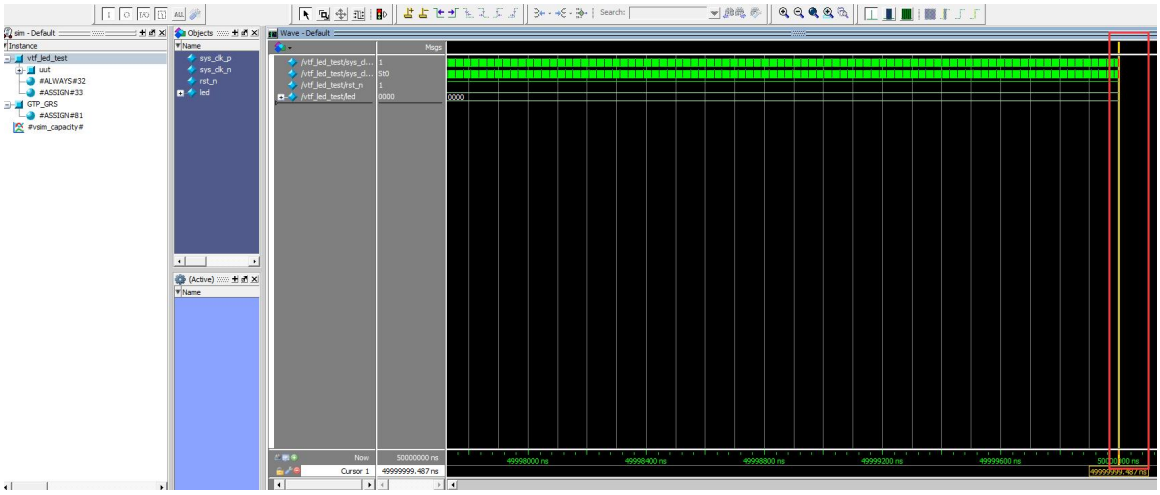


10. 右击仿真文件并在下拉菜单中选择 Run Behavioral Simulation。这里我们做一下行为级的仿真就可以了。

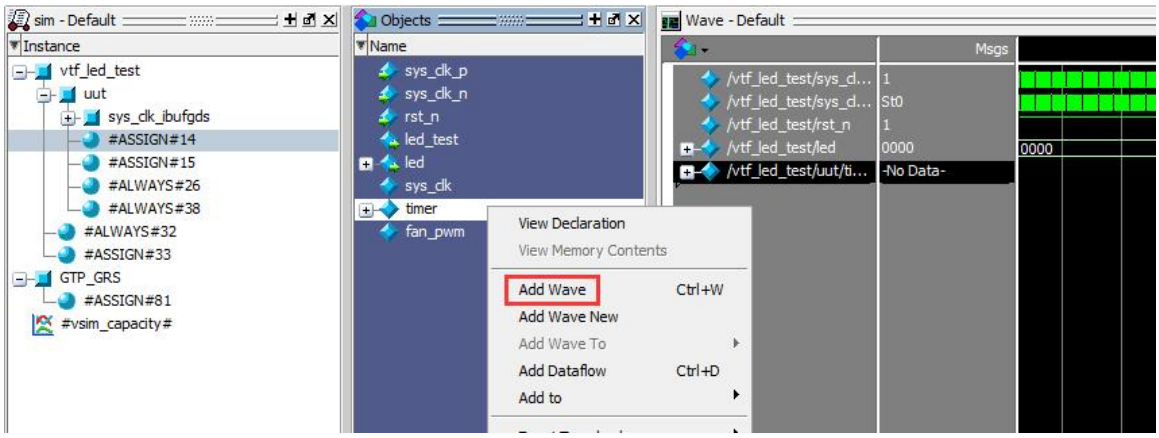


如果没有错误，PDS 会调用 Modelsim 仿真软件开始工作了。

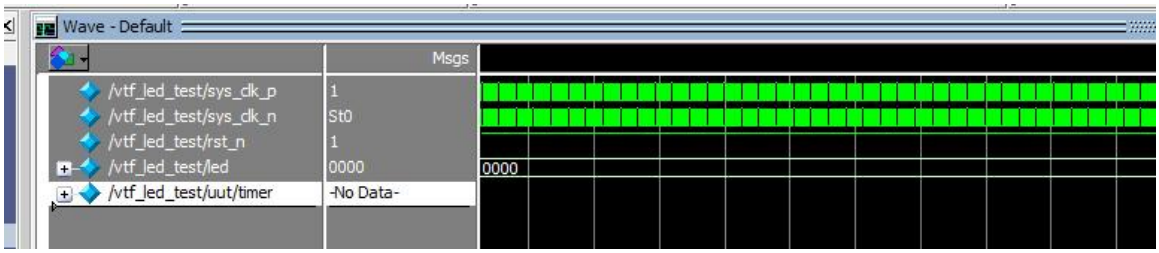
11. 在弹出仿真界面后如下图，界面是仿真软件自动运行到仿真设置的 50ms 的波形。



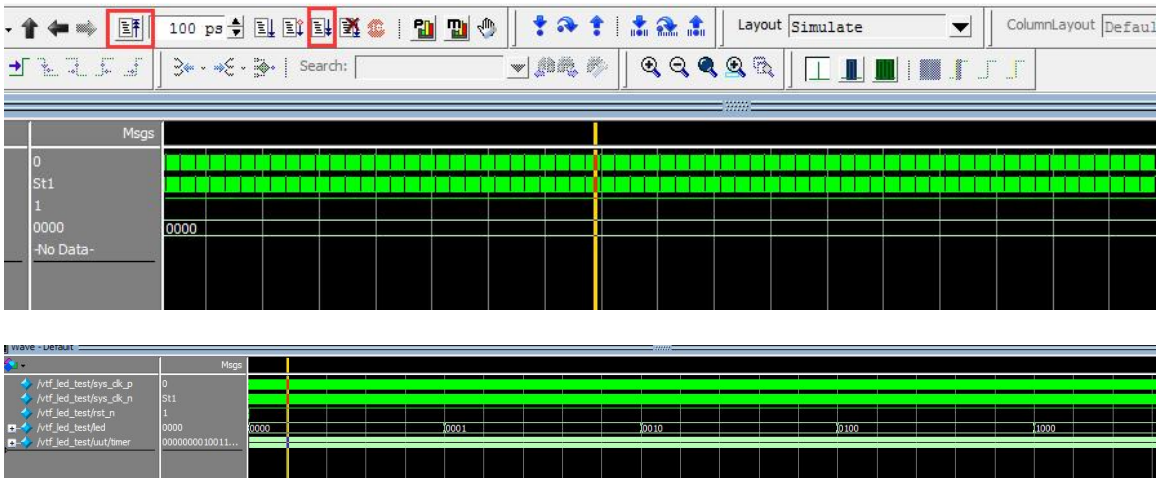
由于 LED[3：0]在程序中设计的状态变化时间长，而仿真又比较耗时，在这里观测 timer[31:0]计数器变化。把它放到 Wave 中观察(点击界面中的 uut， 再右击右侧 timer, 在弹出的下拉菜单里选择 Add Wave)。



添加后 timer 显示在 Wave 的波形界面上，如下图所示。



12. 点击 Restart 按钮复位一下，再点击 Run All 按钮。（需要耐心！！！），可以看到仿真波形与设计相符。



我们可以看到 led 的信号会逐一变 1，说明 LED1~LED4 灯逐个熄灭。

这里为止，我们的第一个项目就圆满完成了，相信您也掌握了 PDS 的 FPGA 开发的整个流程，再也不是那个 FPGA 的门外汉了吧！ 师傅领进门，修行还需要靠本身！PDS 软件的一些技巧的使用和掌握就需要靠大家在长期实践和探索中慢慢熟悉了。

5 附录

led_test.v(verilog 代码)

```
        `timescale 1ns/1ns
module led_test
(
input        sys_clk_p,        //system clock positive
input        sys_clk_n,        //system clock negative
input        rst_n,           //reset ,low active
output        led_test,
output reg[3:0]   led   // LED,use for control the LED signal on board
 );

wire sys_clk;
//define the time counter
reg [31:0]   timer;
assign fan_pwm=1'b0;
assign led_test=1'b1;
GTP_INBUFGDS sys_clk_ibufgds
(
.I                  (sys_clk_p            ),
.IB                 (sys_clk_n            ),
.O                  (sys_clk              )
);

//=============================================================================
// cycle counter:from 0 to 1sec
//=============================================================================
always @(posedge sys_clk or negedge rst_n)
begin
 if (~rst_n)
    timer <= 32'd0;                // when the reset signal valid,time counter clearing
 else if (timer == 32'd199_999_999)   //4 seconds count(200M-1=199999999)
    timer <= 32'd0;                  //count done,clearing the time counter
 else
    timer <= timer + 1'b1;          //timer counter = timer counter + 1
end
//=============================================================================
// LED control
//=============================================================================
always @(posedge sys_clk or negedge rst_n)
begin
 if (~rst_n)
    led <= 4'b0000;               //when the reset signal active
 else if (timer == 32'd49_999_999)   //time counter count to 0.25 sec,LED1 lighten
    led <= 4'b0001;
 else if (timer == 32'd99_999_999)   //time counter count to 0.5 sec,LED2lighten
 begin
    led <= 4'b0010;
  end
```

```
  else if (timer == 32'd149_999_999)   //time counter count to 0.75 sec,LED3 lighten
     led <= 4'b0100;
  else if (timer == 32'd199_999_999)   //time counter count to 1 sec,LED4 lighten
     led <= 4'b1000;
end
     endmodule
```

注意：在定义寄存器时，如果寄存器在 always 块里使用必须定义为 reg 类型，如果仅是用于连线或是直接赋值需定义为 wire 类型，输入信号的类型不能定义为 reg 型，不管是 reg 类型信号还是 wire 类型的信号，定义的寄存器宽度必须满足使用时的需要，但必须稍大于或等于需要使用的位宽。若定义寄存器位宽远远大于使用需求则会浪费资源，如果定义的位宽小于使用需求，则会造成数据位截断，导致程序错误。还有其他信号的类型及用法请大家参考 Verilog 语法教程。